FIG. 2B), and so on, regardless of what other configurations may specify. With the various configurations, updates to applications via new assemblies are possible, but only in a safe, controlled manner, with the machine administrator having the final determination.

[**0056**] Although complete compatibility of a shared assembly should be thoroughly tested, even the smallest change to a shared assembly's code may cause an incompatibility with some applications that consume them. In such an event, in the second alternative mode wherein the publisher configuration data can override the application configuration data, the application configuration file may specify that "safe mode" binding should be applied to the specified assembly. In the safe mode, the publisher configuration resolution stage is avoided, whereby the application operates with no publisher configuration overrides. To use the safe mode, the application configuration **218** (or other system setting, e.g., set by an administrator) can explicitly instruct the operating system to bypass the interpretation of the publisher configuration **220**, whereby the application author (or an administrator) controls the version that is bound to the application. In **FIG. 2**B, this is represented by the wide arrow (labeled "Safe Mode") from block **218** to block **224**. Any administrator policy still may make the final decision, however.

[**0057**] It should be noted that the safe mode bypasses publisher configuration in the second alternative mode, which may affect the version that is used even when the administrator configuration includes an instruction related to a version of that assembly. By way of an example, as shown in **FIG. 2**B, consider the application configuration specifying the safe mode. If the administrator configuration includes an instruction to change version 4.0.0.0 back to version 3.0.0.0, but no others with respect to this assembly, the change to version 3.0.0.0 will not be implemented in the safe mode because the application configuration has specified version 2.0.0.0, whereby the administrator configuration will see version 2.0.0.0 (and not see version 4.0.0.0) to change it. However, if not in the safe mode, the version will be changed to 3.0.0.0 because the publisher configuration will have first changed the version to 4.0.0.0 as described above, whereby the administrator configuration instruction for version 4.0.0.0 will apply and version 3.0.0.0 will be restored. To provide flexibility, a configuration instruction can specify a range of versions to redirect, e.g., change any version in the range from 1.2.3.4 to 5.6.7.8 to version 9.0.0.0.

[**0058**] Note that while two alternative modes are primarily described herein, it can be readily appreciated that other such modes are feasible. For example, it is feasible to have a third alternative mode similar to the first alternative mode described above wherein a publisher configuration is interpreted before an application configuration, but the further including an administrator configuration as in the second alternative mode. Indeed, the present invention is not limited to any particular ordering, number and types of configurations, and so on, but rather contemplates any such combinations and permutations.

[**0059**] As described above, binding starts with a reference, for example, that at least contains the name or other identifier of the assembly. A fully-specified assembly reference (e.g., in the manifest and/or configurations) contains the information necessary to disambiguate one assembly from another. Dependent assembly references, which are constructed at link time, are fully-specified assembly references. However, under some circumstances, it may be desirable to provide only a subset of the assembly identity information, yet still issue a bind. For example, a partially-specified assembly reference may be missing public key, and/or version fields.

[**0060**] Because partially-specified references are ambiguous, the binding process can employ special logic to locate and bind to these assemblies. More particularly, a first step to resolving a partial-specified assembly bind is to search for an assembly in the application directory that satisfies the specified fields in the assembly reference. The application directory is probed first, (as opposed to immediately searching for a matching assembly in the global assembly cache **212**), to provide an application author/deployer with some control over the assembly that is finally retrieved through a partially specified bind request. In other words, because the global assembly cache **212** is a global install location, which can be used by all applications, searching the global assembly cache **212** first may result in an assembly being returned that was not intended by the original author/deployer.

[**0061**] Should the binding process be unable to locate a matching assembly in the application directory, a lookup in the global assembly cache **212** may be performed to attempt to find a matching assembly, (e.g. for strongly-named files in the first alternative mode). If a match is not found in the global assembly cache **212**, the assembly bind may be failed (e.g., in the first alternative mode), or a download can be attempted (e.g., in the second alternative mode). If a matching assembly has been located, the binding process (e.g., the binding/initialization mechanism) reads the manifest data for the assembly and constructs a new, fully-specified assembly reference from this data. Because the assembly reference is now fully-specified, binding configuration can be applied on this reference, as described above, that is, the bind proceeds as if the original assembly reference was fully-specified, with the above-described logic used to specify one assembly file that, if located, satisfies the bind request.

[**0062**] **FIG. 3A** generally represents the interpreting of the manifest and configurations in the first alternative mode, beginning when one (or more) of a set of activation APIs (application programming interfaces) **300** are called in response to a request to run an application program, e.g., the application program **200**. It should be noted that not every assembly used by an application needs to be specified in the application manifest, nor does each assembly manifest have to specify all of its dependencies. For example, the architecture allows for binding to components, as was previously done, so a component (such component files are not usually referred to as assemblies) not specified in a manifest would expect to be found through a search path. In addition, it is feasible for the architecture to allow a default assembly version to be used when a particular version of an assembly is not specified in a manifest. When more than one version of an assembly is available, the default assembly is the most-recent version, however it is possible for an administrator or user to set (e.g., via interaction with the operating system through a dialog box, property sheet or the like) any